**What is Node.js?**

Node.js is a web application framework built on Google Chrome's JavaScript Engine(V8 Engine).

Node.js comes with runtime environment on which a Javascript based script can be interpreted and executed (It is analogus to JVM to JAVA byte code). This runtime allows to execute a JavaScript code on any machine outside a browser. Because of this runtime of Node.js, JavaScript is now can be executed on server as well.

Node.js also provides a rich library of various javascript modules which eases the developement of web application using Node.js to great extents.

Node.js = Runtime Environment + JavaScript Library

What do you mean by Asynchronous API?

All APIs of Node.js library are aynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.

What are the benefits of using Node.js?

Following are main benefits of using Node.js

* **Asynchronous and Event Driven**All APIs of Node.js library are asynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.
* **Very Fast** Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* **Single Threaded but highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps server to respond in a non-bloking ways and makes server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and same program can services much larger number of requests than traditional server like Apache HTTP Server.
* **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.

Is it free to use Node.js?

Yes! Node.js is released under the [MIT license](https://raw.githubusercontent.com/joyent/node/v0.12.0/LICENSE) and is free to use.

Is Node a single threaded application?

Yes! Node uses a single threaded model with event looping.

What is REPL in context of Node?

REPL stands for Read Eval Print Loop and it represents a computer environment like a window console or unix/linux shell where a command is entered and system responds with an output. Node.js or Node comes bundled with a REPL environment. It performs the following desired tasks.

* **Read** − Reads user's input, parse the input into JavaScript data-structure and stores in memory.
* **Eval** − Takes and evaluates the data structure
* **Print** − Prints the result
* **Loop** − Loops the above command until user press ctrl-c twice.

Can we evaluate simple expression using Node REPL

Yes.

What is the difference of using var and not using var in REPL while dealing with variables?

Use variables to store values and print later. if var keyword is not used then value is stored in the variable and printed. Whereas if var keyword is used then value is stored but not printed. You can use both variables later.

What is the use of Underscore variable in REPL?

Use **\_** to get the last result.

C:\Nodejs\_WorkSpace>node> var x = 10undefined> var y = 20undefined> x + y30> var sum = \_undefined> console.log(sum)30undefined>

What is npm?

npm stands for Node Package Manager. npm provides following two main functionalities:

* Online repositories for node.js packages/modules which are searchable on [search.nodejs.org](http://search.nodejs.org/)
* Command line utility to install packages, do version management and dependency management of Node.js packages.

What is global installation of dependencies?

Globally installed packages/dependencies are stored in **<user-directory>**/npm directory. Such dependencies can be used in CLI (Command Line Interface) function of any node.js but can not be imported using require() in Node application directly. To install a Node project globally use -g flag.

C:\Nodejs\_WorkSpace>npm install express -g

What is local installation of dependencies?

By default, npm installs any dependency in the local mode. Here local mode refers to the package installation in node\_modules directory lying in the folder where Node application is present. Locally deployed packages are accessible via require(). To install a Node project locally following is the syntax.

C:\Nodejs\_WorkSpace>npm install express

How to check the already installed dependencies which are globally installed using npm?

Use the following command −

C:\Nodejs\_WorkSpace>npm ls -g

What is Package.json?

package.json is present in the root directory of any Node application/module and is used to define the properties of a package.

Name some of the attributes of package.json?

Following are the attributes of Package.json

* **name** − name of the package
* **version** − version of the package
* **description** − description of the package
* **homepage** − homepage of the package
* **author** − author of the package
* **contributors** − name of the contributors to the package
* **dependencies** − list of dependencies. npm automatically installs all the dependencies mentioned here in the node\_module folder of the package.
* **repository** − repository type and url of the package
* **main** − entry point of the package
* **keywords** − keywords

How to uninstall a dependency using npm?

Use following command to uninstall a module.

C:\Nodejs\_WorkSpace>npm uninstall dependency-name

How to update a dependency using npm?

Update package.json and change the version of the dependency which to be updated and run the following command.

C:\Nodejs\_WorkSpace>npm update

What is Callback?

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All APIs of Node are written is such a way that they supports callbacks. For example, a function to read a file may start reading file and return the control to execution environment immidiately so that next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process high number of request without waiting for any function to return result.

What is a blocking code?

If application has to wait for some I/O operation in order to complete its execution any further then the code responsible for waiting is known as blocking code.

How Node prevents blocking code?

By providing callback function. Callback function gets called whenever corresponding event triggered.

What is Event Loop?

Node js is a single threaded application but it support concurrency via concept of event and callbacks. As every API of Node js are asynchronous and being a single thread, it uses async function calls to maintain the concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever any task get completed, it fires the corresponding event which signals the event listener function to get executed.

What is Event Emmitter?

EventEmitter class lies in **events** module. It is accessibly via following syntax −

//import events modulevar events = require('events');//create an eventEmitter objectvar eventEmitter = new events.EventEmitter();

When an EventEmitter instance faces any error, it emits an 'error' event. When new listener is added, 'newListener' event is fired and when a listener is removed, 'removeListener' event is fired.

EventEmitter provides multiple properties like **on** and **emit**. **on** property is used to bind a function with the event and **emit** is used to fire an event.

What is purpose of Buffer class in Node?

Buffer class is a global class and can be accessed in application without importing buffer module. A Buffer is a kind of an array of integers and corresponds to a raw memory allocation outside the V8 heap. A Buffer cannot be resized.

What is Piping in Node?

Piping is a mechanism to connect output of one stream to another stream. It is normally used to get data from one stream and to pass output of that stream to another stream. There is no limit on piping operations. Consider the above example, where we've read test.txt using readerStream and write test1.txt using writerStream. Now we'll use the piping to simplify our operation or reading from one file and writing to another file.

Which module is used for file based operations?

fs module is used for file based operations.

var fs = require("fs")

Which module is used for buffer based operations?

buffer module is used for buffer based operations.

var buffer = require("buffer")

Which module is used for web based operations?

http module is used for web based operations.

var http = require("http")

fs module provides both synchronous as well as asynchronous methods.

true.

What is difference between synchronous and asynchronous method of fs module?

Every method in fs module have synchronous as well as asynchronous form. Asynchronous methods takes a last parameter as completion function callback and first parameter of the callback function is error. It is preferred to use asynchronous method instead of synchronous method as former never block the program execution where the latter one does.

Name some of the flags used in read/write operation on files.

flags for read/write operations are following:

* **r** − Open file for reading. An exception occurs if the file does not exist.
* **r+** − Open file for reading and writing. An exception occurs if the file does not exist.
* **rs** − Open file for reading in synchronous mode. Instructs the operating system to bypass the local file system cache. This is primarily useful for opening files on NFS mounts as it allows you to skip the potentially stale local cache. It has a very real impact on I/O performance so don't use this flag unless you need it. Note that this doesn't turn fs.open() into a synchronous blocking call. If that's what you want then you should be using fs.openSync()
* **rs+** − Open file for reading and writing, telling the OS to open it synchronously. See notes for 'rs' about using this with caution.
* **w** − Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
* **wx** − Like 'w' but fails if path exists.
* **w+** − Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
* **wx+** − Like 'w+' but fails if path exists.
* **a** − Open file for appending. The file is created if it does not exist.
* **ax** − Like 'a' but fails if path exists.
* **a+** − Open file for reading and appending. The file is created if it does not exist.
* **ax+'** − Like 'a+' but fails if path exists.

What are streams?

Streams are objects that let you read data from a source or write data to a destination in continuous fashion.

How many types of streams are present in Node.

In Node.js, there are four types of streams.

* **Readable** − Stream which is used for read operation.
* **Writable** − Stream which is used for write operation.
* **Duplex** − Stream which can be used for both read and write operation.
* **Transform** − A type of duplex stream where the output is computed based on input.

Name some of the events fired by streams.

Each type of Stream is an **EventEmitter** instance and throws several events at different instance of times. For example, some of the commonly used events are:

* **data** − This event is fired when there is data is available to read.
* **end** − This event is fired when there is no more data to read.
* **error** − This event is fired when there is any error receiving or writing data.
* **finish** − This event is fired when all data has been flushed to underlying system

What is Chaining in Node?

Chanining is a mechanism to connect output of one stream to another stream and create a chain of multiple stream operations. It is normally used with piping operations.

How will you open a file using Node?

Following is the syntax of the method to open a file in asynchronous mode:

fs.open(path, flags[, mode], callback)

### Parameters

Here is the description of the parameters used:

* **path** − This is string having file name including path.
* **flags** − Flag tells the behavior of the file to be opened. All possible values have been mentioned below.
* **mode** − This sets the file mode (permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.
* **callback** − This is the callback function which gets two arguments (err, fd).

How will you read a file using Node?

Following is the syntax of one of the methods to read from a file:

fs.read(fd, buffer, offset, length, position, callback)

This method will use file descriptor to read the file, if you want to read file using file name directly then you should use another method available.

### Parameters

Here is the description of the parameters used −

* **fd** − This is the file descriptor returned by file fs.open() method.
* **buffer** − This is the buffer that the data will be written to.
* **offset** − This is the offset in the buffer to start writing at.
* **length** − This is an integer specifying the number of bytes to read.
* **position** − This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.
* **callback** − This is the callback function which gets the three arguments, (err, bytesRead, buffer).

How will you write a file using Node?

Following is the syntax of one of the methods to write into a file:

fs.writeFile(filename, data[, options], callback)

This method will over-write the file if file already exists. If you want to write into an existing file then you should use another method available.

### Parameters

Here is the description of the parameters used:

* **path** − This is string having file name including path.
* **data** − This is the String or Buffer to be written into the file.
* **options** − The third parameter is an object which will hold {encoding, mode, flag}. By default encoding is utf8, mode is octal value 0666 and flag is 'w'
* **callback** − This is the callback function which gets a single parameter err and used to to return error in case of any writing error.

How will you close a file using Node?

Following is the syntax of one of the methods to close an opened file:

fs.close(fd, callback)

### Parameters

Here is the description of the parameters used:

* **fd** − This is the file descriptor returned by file fs.open() method.
* **callback** − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you get information about a file using Node?

Following is the syntax of the method to get the information about a file:

fs.stat(path, callback)

### Parameters

Here is the description of the parameters used:

* **path** − This is string having file name including path.
* **callback** − This is the callback function which gets two arguments (err, stats) where **stats** is an object of fs.Stats type which is printed below in the example.

How will you truncate a file using Node?

Following is the syntax of the method to truncate an opened file −

fs.ftruncate(fd, len, callback)

### Parameters

Here is the description of the parameters used:

* **fd** − This is the file descriptor returned by file fs.open() method.
* **len** − This is the length of the file after which file will be truncated.
* **callback** − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you delete a file using Node?

Following is the syntax of the method to delete a file −

fs.unlink(path, callback)

### Parameters

Here is the description of the parameters used:

* **path** − This is the file name including path.
* **callback** − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you create a directory?

Following is the syntax of the method to create a directory:

fs.mkdir(path[, mode], callback)

### Parameters

Here is the description of the parameters used:

* **path** − This is the directory name including path.
* **mode** − This is the directory permission to be set. Defaults to 0777.
* **callback** − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you delete a directory?

Following is the syntax of the method to remove a directory:

fs.rmdir(path, callback)

### Parameters

Here is the description of the parameters used:

* **path** − This is the directory name including path.
* **callback** − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you read a directory?

Following is the syntax of the method to read a directory:

fs.readdir(path, callback)

### Parameters

Here is the description of the parameters used:

* **path** − This is the directory name including path.
* **callback** − This is the callback function which gets two arguments (err, files) where files is an array of the names of the files in the directory excluding '.' and '..'.

What is the purpose of \_\_filename variable?

The \_\_filename represents the filename of the code being executed. This is the resolved absolute path of this code file. For a main program this is not necessarily the same filename used in the command line. The value inside a module is the path to that module file.

What is the purpose of \_\_dirname variable?

The \_\_dirname represents the name of the directory that the currently executing script resides in.

What is the purpose of setTimeout function?

The setTimeout(cb, ms) global function is used to run callback cb after at least ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days.

This function returns an opaque value that represents the timer which can be used to clear the timer.

What is the purpose of clearTimeout function?

The clearTimeout( t ) global function is used to stop a timer that was previously created with setTimeout(). Here t is the timer returned by setTimeout() function.

What is the purpose of setInterval function?

The setInterval(cb, ms) global function is used to run callback cb repeatedly after at least ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days.

This function returns an opaque value that represents the timer which can be used to clear the timer using the function clearInterval(t).

What is the purpose of console object?

console object is used to Used to print information on stdout and stderr.

What is the purpose of process object?

process object is used to get information on current process. Provides multiple events related to process activities.

**Q1What is Node.js? What is it used for?**

Node.js is a run-time JavaScript environment built on top of Chrome's V8 engine. It uses an event-driven, non-blocking I/O model. It is lightweight and so efficient. Node.js has a package ecosystem called **npm**.

Node.js can be used to build different types of applications such as web application, real-time chat application, REST API server etc. However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET. Node.js was developed by Ryan Dahl in 2009.

**Q2What is Event-driven programming?**

Event-driven programming is building our application based on and respond to events. When an event occurs, like click or keypress, we are running a callback function which is registered to the element for that event.

Event driven programming follows mainly a publish-subscribe pattern.

**function** **addToCart**(productId){  
 event.send("cart.add", {id: productId});  
}  
  
event.on("cart.add", **function**(event){  
 show("Adding product " + event.id);  
});

**Q3What is *Event loop* in Node.js work? And How does it work?**

The *Event loop* handles all async callbacks. Node.js (or JavaScript) is a single-threaded, event-driven language. This means that we can attach listeners to events, and when a said event fires, the listener executes the callback we provided.

Whenever we are call setTimeout, http.get and fs.readFile, Node.js runs this operations and further conitnue to run other code without waiting for the output. When the operation is finished, it receives the output and runs our callback function.

So all the callback functions are queued in an loop, and will run one-by-one when the response has been received.

**Q4What is REPL in Node.js?**

REPL means Read-Eval-Print-Loop. It is a virtual environment that comes with Node.js. We can quickly test our JavaScript code in the Node.js REPL environment.

To launch the REPL in Node.js, just opne the command prompt and type node. It will change the prompt to > in Windows and MAC.

Now we can type and run our JavaScript easily. For example, if we type 10 + 20, it will print 30 in the next line.

**Q5What is the purpose of module.exports in Node.js?**

A module encapsulates related code into a single unit of code. This can be interpreted as moving all related functions into a file. Imagine that we created a file called greetings.js and it contains the following two functions:

module.exports = {  
 sayHelloInEnglish: **function**() {  
 **return** "HELLO";  
 },  
  
 sayHelloInSpanish: **function**() {  
 **return** "Hola";  
 }  
};

In the above code, module.exports exposes two functions to the outer world. We can import them in another file as follow:

**var** greetings = require("./greetings.js");  
  
greetings.sayHelloInEnglish(); // Hello  
  
greetings.sayHelloInSpanish(); //Hola

**Q6What is require in Node.js? How will you load external files and libraries in Node.js?**

require function is used to load external files and modules that exist in separate files. The basic functionality of require is that it reads the a JavaScript files, execute the file and then return the exports object. For example, consider the below code exist in example.js:

exports.sayHello = **function**(name) {  
 console.log("Hello " + name + "!");  
}

In another file, we can load this function with require.

**var** example = require("./example");  
example("World"); //Hello World!

Modules are cached after the first time they are loaded. This means that every call to require('example') will get exactly the same object returned, if it would resolve to the same file.

**Q7What is the difference between Asynchronous and Non-blocking?**

Asynchronous literally means not synchronous. We are making HTTP requests which are asynchronous, means we are not waiting for the server response. We continue with other block and respond to the server response when we received.

The term Non-Blocking is widely used with IO. For example non-blocking read/write calls return with whatever they can do and expect caller to execute the call again. Read will wait until it has some data and put calling thread to sleep.

**Q8What is Tracing in Node.js?**

Tracing provides a mechanism to collect tracing information generated by V8, Node core and userspace code in a log file. Tracing can be enabled by passing the --trace-events-enabled flag when starting a Node.js application.

**node** **--trace-events-enabled** **--trace-event-categories** **v8**,**node** **server**.js

The set of categories for which traces are recorded can be specified using the --trace-event-categories flag followed by a list of comma separated category names. By default the node and v8 categories are enabled.

Running Node.js with tracing enabled will produce log files that can be opened in the chrome://tracing tab of Chrome.

**Q9How will you debug an application in Node.js?**

Node.js includes a debugging utility called debugger. To enable it start the Node.js with the debug argument followed by the path to the script to debug.

Inserting the statement debugger; into the source code of a script will enable a breakpoint at that position in the code:

x = 5;  
setTimeout(() => {  
 **debugger**;  
 console.log('world');  
}, 1000);

**Q10Difference between setImmediate() vs setTimeout()**

setImmediate() and setTimeout() are similar, but behave in different ways depending on when they are called.

* setImmediate() is designed to execute a script once the current poll (event loop) phase completes.
* setTimeout() schedules a script to be run after a minimum threshold in ms has elapsed.

The order in which the timers are executed will vary depending on the context in which they are called. If both are called from within the main module, then timing will be bound by the performance of the process.

**Q11What is process.nextTick()**

setImmediate() and setTimeout() are based on the event loop. But process.nextTick() technically not part of the event loop. Instead, the nextTickQueue will be processed after the current operation completes, regardless of the current phase of the event loop.

Thus, any time you call process.nextTick() in a given phase, all callbacks passed to process.nextTick() will be resolved before the event loop continues.

**Q12What is package.json? What is it used for?**

This file holds various metadata information about the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

Some of the fields are: name, name, description, author and dependencies.

When someone installs our project through npm, all the dependencies listed will be installed as well. Additionally, if someone runs npm install in the root directory of our project, it will install all the dependencies to ./node\_modules directory.

**Q13What is libuv?**

libuv is a multi-platform support library with a focus on asynchronous I/O. It was primarily developed for use by Node.js, but it’s also used by Luvit, Julia, pyuv, and others.

When the node.js project began in 2009 as a JavaScript environment decoupled from the browser, it is using Google’s V8 and Marc Lehmann’s libev, node.js combined a model of I/O – evented – with a language that was well suited to the style of programming; due to the way it had been shaped by browsers. As node.js grew in popularity, it was important to make it work on Windows, but libev ran only on Unix. libuv was an abstraction around libev or IOCP depending on the platform, providing users an API based on libev. In the node-v0.9.0 version of libuv libev was removed.

Some of the features of libuv are:

* Full-featured event loop backed by epoll, kqueue, IOCP, event ports.
* Asynchronous TCP and UDP sockets
* Asynchronous file and file system operations
* Child processes
* File system events

**Q14What are some of the most popular modules of Node.js?**

There are many most popular, most starred or most downloaded modules in Node.js. Some of them are:

* express
* async
* browserify
* socket.io
* bower
* gulp
* grunt

**Q15What is express.js?**

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Express is a light-weight web application framework to help organize our web application into a MVC architecture on the server side. We can use a variety of choices for your templating language like EJS, Jade.

Express.js basically helps you manage everything, from routes, to handling requests, response and views. An example of an Express.js routing is as follow:

**var** express = require('express')  
var app = express()  
  
// respond with "hello world" when a GET request is made to the homepage  
app.get('/', function (req, res) {  
 res.send('hello world')  
})

**Q16What is middlewares? What is the concept of middlewares in Express.js?**

Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. The next middleware function is commonly denoted by a variable named next.

Middleware functions can perform the following tasks:

* Execute any code.
* Make changes to the request and the response objects.
* End the request-response cycle.
* Call the next middleware function in the stack.

Bind application-level middleware to an instance of the app object by using the app.use() and app.METHOD() functions, where METHOD is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST) in lowercase.

**var** app = express()  
  
app.use(**function** (req, res, next) {  
 console.log('Time:', Date.now())  
 next()  
})

**Q17What is EventEmitter in Node.js?**

All objects that emit events are instances of the EventEmitter class. These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.

When the EventEmitter object emits an event, all of the functions attached to that specific event are called *synchronously*.

**const** events = require('events');  
const eventEmitter = **new** events.EventEmitter();  
  
let myEvent = **function** **ringBell**() {  
 console.log('Event is emitted');  
}  
  
eventEmitter.on('emitEvent', myEvent);  
  
eventEmitter.emit('emitEvent');

**Q18What is Streams in Node.js?**

Streams are pipes that let you easily read data from a source and pipe it to a destination. Simply put, a stream is nothing but an EventEmitter and implements some specials methods. Depending on the methods implemented, a stream becomes Readable, Writable, or Duplex (both readable and writable).

For example, if we want to read data from a file, the best way to do it from a stream is to listen to data event and attach a callback. When a chunk of data is available, the readable stream emits a data event and your callback executes. Take a look at the following snippet:

**var** fs = require('fs');  
var readableStream = fs.createReadStream('textFile.txt');  
var fileData = '';  
  
readableStream.on('data', **function**(chunk) {  
 data += chunk;  
});  
  
readableStream.on('end', **function**() {  
 console.log(data);  
});

Types of streams are: Readable, Writable, Duplex and Transform.

**Q19What is Child Process and Cluster? What are the difference?**

Child Process in Node.js is used to run a child process under Node.js. There are two methods to do that: exec and spawn. The example for exec is:

**var** exec = require('child\_process').exec;  
exec('node -v', **function**(error, stdout, stderr) {  
 console.log('stdout: ' + stdout);  
 console.log('stderr: ' + stderr);  
 **if** (error !== null) {  
 console.log('exec error: ' + error);  
}  
});

We are passing our command **as** our first argument **of** the exec and expect three response **in** the callback. *stdout* is the output we could expect **from** the execution.

Cluster is used to split a single process into multiple processes or *workers*, **in** Node.js terminology. This can be achieved through a cluster module. The cluster module allows you to create child processes (workers), which share all the server ports **with** the main Node process (master).

A cluster is a pool **of** similar workers running under a parent Node process. Workers are spawned using the fork() method **of** the child\_processes module. Cluster is multiple processes to scale on multi cores.

**Q20How can we avoid Callback Hell in Node.js?**

Callback hell refers to heavily nested callbacks that have become unreadable and hard to debug. We can use async module.

Async is a utility module which provides straight-forward, powerful functions for working with asynchronous JavaScript. Consider we have two functions which runs sequentially:

**async**.waterfall([firstFunction, secondFunction], **function**() {  
 console.log('done')  
});

**Q21What is the difference between Node.js vs Ajax?**

The difference between Node.js and Ajax is that, Ajax (short for Asynchronous JavaScript and XML) is a client side technology, often used for updating the contents of the page without refreshing it. While,Node.js is Server Side JavaScript, used for developing server software. Node.js does not execute in the browser but by the server.

**Q22What is File System in Node.js?**

Node.js provides a File system module to read and write data to a file in the memory. This module comes with Node.js by default in the name of fs. To read a file:

**var** fs = require("fs");  
  
fs.readFile("foo.txt", "utf8", **function**(err, data) {  
 console.log(data);  
});

**Q23What is the difference between readFile vs createReadStream in Node.js?**

**readFile** - is for asynchronously reads the entire contents of a file. It will read the file completely into memory before making it available to the User. readFileSync is synchronous version of readFile.

**createReadStream** - It will read the file in chunks of the default size 64 kb which is specified before hand.

**Q24How to support multi-processor platforms in Node.js?**

Since Node.js is by default a single thread application, it will run on a single processor core and will not take full advantage of multiple core resources. However, Node.js provides support for deployment on multiple-core systems, to take greater advantage of the hardware. The Cluster module is one of the core Node.js modules and it allows running multiple Node.js worker processes that will share the same port.

**Q25How to install a node module locally and globally? How to add them into package.json?**

By default, Node.js installs the module locally in the ./node\_modules directory. To install a module globally and to make them available everywhere, we need to include the -g flag.

npm install express -g

To save the installed module to the package.json under dependencies, add --save flag with the package name. To add under deveDependencies add --save-dev.

**Q26What are Global objects in Node.js?**

Node.js has some Global objects which are available in all modules. Some of these objects are not actually in global scope but in the module scope.

* **\_\_dirname**: The directory name of the current module. This the same as the path.dirname() of the \_\_filename.
* **\_\_filename**: The file name of the current module. This is the resolved absolute path of the current module file.
* **exports**: A reference to the module.exports that is shorter to type. exportsis not actually a global but rather local to each module.
* **require**: To require modules. require is not actually a global but rather local to each module.

**Q27What is crypto in Node.js? How do you cipher the secured information in Node.js?**

The crypto module in Node.js provides cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign and verify functions.

**var** crypto = require('crypto');  
  
var secret = 'abcdefg';  
var hash = crypto.createHmac('sha256', secret)  
 .update('I love cupcakes')  
 .digest('hex');  
console.log(hash);  
  
 // c0fa1bc00531bd78ef38c628449c5102aeabd49b5dc3a2a516ea6ea959d6658e

**Q28What is zlib? How do you compress a file in Node.js?**

The zlib module provides compression functionality implemented using Gzip and Deflate/Inflate.

**var** zlib = require('zlib');  
  
var gzip = zlib.createGzip();  
var fs = require('fs');  
var inp = fs.createReadStream('input.txt');  
  
var out = fs.createWriteStream('input.txt.gz');  
  
inp.pipe(gzip).pipe(out);

Calling .flush() on a compression stream will make zlib return as much output as currently possible. This may come at the cost of degraded compression quality, but can be useful when data needs to be available as soon as possible.

**Q29What is Buffer in Node.js?**

Buffers are instances of the Buffer class in node, which is designed to handle raw binary data. Each buffer corresponds to some raw memory allocated outside V8. Buffers act somewhat like arrays of integers, but aren't resizable and have a whole bunch of methods specifically for binary data.

Pure JavaScript does not handle straight binary data very well. This is fine on the browser, where most data is in the form of strings. However, Node.js servers have to also deal with TCP streams and reading and writing to the filesystem, both which make it necessary to deal with purely binary streams of data. Below are the ways to create new buffers:

**var** bufferOne = **new** Buffer(8);  
 // Uninitalized buffer and contains 8 bytes  
  
var bufferTwo = new Buffer([ 8, 6, 7]);  
 //This initializes the buffer to the contents of this array. The array are integers representing bytes.  
  
var bufferThree = new Buffer("This is a string", "utf-8")  
 // This initializes the buffer to a binary encoding of the first string as specified by the second argument (in this case, utf-8).

**Q30How to test your code in Node.js?**

The assert module in Node.js provides easy way of writing test in a limited ways. It provides no feedback when running your tests unless one fails. It provides 11 methods to test our code to check it is working as we expect. Some of them are: assert.equal, assert.deepEqual, assert.throws.

assert.equal(2 + 2, 4, 'two plus two is four');

Other than this, we can use testing frameworks like, mocha, chai and karma.

**What is node.js?**

Node.js is a Server side scripting which is used to build scalable programs. Its multiple advantages over other server side languages, the prominent being non-blocking I/O.

**2) How node.js works?**

Node.js works on a v8 environment, it is a virtual machine that utilizes JavaScript as its scripting language and achieves high output via non-blocking I/O and single threaded event loop.

**3) What do you mean by the term I/O ?**

I/O is the shorthand for input and output, and it will access anything outside of your application. It will be loaded into the machine memory to run the program, once the application is started.



**4) What does event-driven programming mean?**

In computer programming, event driven programming is a programming paradigm in which the flow of the program is determined by events like messages from other programs or threads. It is an application architecture technique divided into two sections 1) Event Selection 2) Event Handling

**5) Where can we use node.js?**

Node.js can be used for the following purposes

a) Web applications ( especially real-time web apps )

b) Network applications

c) Distributed systems

d) General purpose applications

**6) What is the advantage of using node.js?**

a) It provides an easy way to build scalable network programs

b) Generally fast

c) Great concurrency

d) Asynchronous everything

e) Almost never blocks

**7) What are the two types of API functions in Node.js ?**

The two types of API functions in Node.js are

a) Asynchronous, non-blocking functions

b) Synchronous, blocking functions

**8) What is control flow function?**

A generic piece of code which runs in between several asynchronous function calls is known as control flow function.

**9) Explain the steps how “Control Flow” controls the functions calls?**

a) Control the order of execution

b) Collect data

c) Limit concurrency

d) Call the next step in program

**10) Why Node.js is single threaded**?

For async processing, Node.js was created explicitly as an experiment. It is believed that more performance and scalability can be achieved by doing async processing on a single thread under typical web loads than the typical thread based implementation.

**11) Does node run on windows?**

Yes – it does. Download the MSI installer from <http://nodejs.org/download/>

**12) Can you access DOM in node?**

No, you cannot access DOM in node.

**13) Using the event loop what are the tasks that should be done asynchronously?**

a) I/O operations

b) Heavy computation

c) Anything requiring blocking

**14) Why node.js is quickly gaining attention from JAVA programmers?**

Node.js is quickly gaining attention as it is a loop based server for JavaScript. Node.js gives user the ability to write the JavaScript on the server, which has access to things like HTTP stack, file I/O, TCP and databases.

**15) What are the two arguments that async.queue takes?**

The two arguments that async.queue takes

a) Task function

b) Concurrency value

**16) What is an event loop in Node.js ?**

To process and handle external events and to convert them into callback invocations an event loop is used. So, at I/O calls, node.js can switch from one request to another .

**17) Mention the steps by which you can async in Node.js?**

By following steps you can async Node.js

a) First class functions

b) Function composition

c) Callback Counters

d) Event loops

**18) What are the pros and cons of Node.js?**

**Pros:**

a) If your application does not have any CPU intensive computation, you can build it in Javascript top to bottom, even down to the database level if you use JSON storage object DB like MongoDB.

b) Crawlers receive a full-rendered HTML response, which is far more SEO friendly rather than a single page application or a websockets app run on top of Node.js.

**Cons:**

a) Any intensive CPU computation will block node.js responsiveness, so a threaded platform is a better approach.  
b) Using relational database with Node.js is considered less favourable

**19) How Node.js overcomes the problem of blocking of I/O operations?**

Node.js solves this problem by putting the event based model at its core, using an event loop instead of threads.

**20) What is the difference between Node.js vs Ajax?**

The difference between Node.js and Ajax is that, Ajax (short for Asynchronous Javascript and XML) is a client side technology, often used for updating the contents of the page without refreshing it. While,Node.js is Server Side Javascript, used for developing server software. Node.js does not execute in the browser but by the server.

**21) What are the Challenges with Node.js ?**

Emphasizing on the technical side, it’s a bit of challenge in Node.js to have one process with one thread to scale up on multi core server.

**22)**  **What does it mean “non-blocking” in node.js?**

In node.js “non-blocking” means that its IO is non-blocking. Node uses “libuv” to handle its IO in a platform-agnostic way. On windows, it uses completion ports for unix it uses epoll or kqueue etc. So, it makes a non-blocking request and upon a request, it queues it within the event loop which call the JavaScript ‘callback’ on the main JavaScript thread.

**23) What is the command that is used in node.js to import external libraries?**

Command “require” is used for importing external libraries, for example, “var http=require (“http”)”. This will load the http library and the single exported object through the http variable.

**24) Mention the framework most commonly used in node.js?**

“Express” is the most common framework used in node.js

**25) What is ‘Callback’ in node.js?**

Callback function is used in node.js to deal with multiple requests made to the server. Like if you have a large file which is going to take a long time for a server to read and if you don’t want a server to get engage in reading that large file while dealing with other requests, call back function is used. Call back function allows the server to deal with pending request first and call a function when it is finished.

**Refer our** [**Node.js Tutorials**](http://www.guru99.com/node-js-tutorial.html?utm_source=crosslinking&utm_medium=referral&utm_campaign=click) **for an extra edge in your interview.**

### Q1: Provide some example of config file separation for dev and prod environments

Topic: **Node.js**  
**Difficulty: ⭐⭐⭐⭐**

A perfect and flawless configuration setup should ensure:

* keys can be read from file AND from environment variable
* secrets are kept outside committed code
* config is hierarchical for easier findability

Consider the following config file:

var config = { production: { mongo : { billing: '\*\*\*\*' } }, default: { mongo : { billing: '\*\*\*\*' } }}exports.get = function get(env) { return config[env] || config.default;}

And it's usage:

const config = require('./config/config.js').get(process.env.NODE\_ENV);const dbconn = mongoose.createConnection(config.mongo.billing);

🔗**Source:** [github.com/i0natan/nodebestpractices](https://github.com/i0natan/nodebestpractices/blob/master/sections/projectstructre/configguide.md)

### Q2: What are the timing features of Node.js?

Topic: **Node.js**  
**Difficulty: ⭐⭐⭐⭐**

The Timers module in Node.js contains functions that execute code after a set period of time.

* **setTimeout/clearTimeout** - can be used to schedule code execution after a designated amount of milliseconds
* **setInterval/clearInterval** - can be used to execute a block of code multiple times
* **setImmediate/clearImmediate** - will execute code at the end of the current event loop cycle
* **process.nextTick** - used to schedule a callback function to be invoked in the next iteration of the Event Loop

function cb(){ console.log('Processed in next iteration');}process.nextTick(cb);console.log('Processed in the first iteration');

Output:

Processed in the first iterationProcessed in next iteration

🔗**Source:** [github.com/jimuyouyou](https://github.com/jimuyouyou/node-interview-questions)

### Q3: Explain what is Reactor Pattern in Node.js?

Topic: **Node.js**  
**Difficulty: ⭐⭐⭐⭐⭐**

**Reactor Pattern** is an idea of non-blocking I/O operations in Node.js. This pattern provides a handler(in case of Node.js, a *callback function*) that is associated with each I/O operation. When an I/O request is generated, it is submitted to a *demultiplexer*.

This *demultiplexer* is a notification interface that is used to handle concurrency in non-blocking I/O mode and collects every request in form of an event and queues each event in a queue. Thus, the demultiplexer provides the *Event Queue*.

At the same time, there is an Event Loop which iterates over the items in the Event Queue. Every event has a callback function associated with it, and that callback function is invoked when the Event Loop iterates.

🔗**Source:** [hackernoon.com](https://hackernoon.com/the-node-js-system-51090c35dddc)

### Q4: What is LTS releases of Node.js why should you care?

Topic: **Node.js**  
**Difficulty: ⭐⭐⭐⭐⭐**

An **LTS(Long Term Support)** version of Node.js receives all the critical bug fixes, security updates and performance improvements.

LTS versions of Node.js are supported for at least 18 months and are indicated by even version numbers (e.g. 4, 6, 8). They're best for production since the LTS release line is focussed on stability and security, whereas the *Current* release line has a shorter lifespan and more frequent updates to the code. Changes to LTS versions are limited to bug fixes for stability, security updates, possible npm updates, documentation updates and certain performance improvements that can be demonstrated to not break existing applications.

🔗**Source:** [github.com/i0natan/nodebestpractices](https://github.com/i0natan/nodebestpractices/blob/master/sections/production/LTSrelease.md)

### Q5: Why should you separate Express 'app' and 'server'?

Topic: **Node.js**  
**Difficulty: ⭐⭐⭐⭐⭐**

Keeping the API declaration separated from the network related configuration (port, protocol, etc) allows testing the API in-process, without performing network calls, with all the benefits that it brings to the table: fast testing execution and getting coverage metrics of the code. It also allows deploying the same API under flexible and different network conditions. Bonus: better separation of concerns and cleaner code.

API declaration, should reside in app.js:

var app = express();app.use(bodyParser.json());app.use("/api/events", events.API);app.use("/api/forms", forms);

Server network declaration, should reside in /bin/www:

var app = require('../app');var http = require('http');/\*\* \* Get port from environment and store in Express. \*/var port = normalizePort(process.env.PORT || '3000');app.set('port', port);/\*\* \* Create HTTP server. \*/var server = http.createServer(app);

🔗**Source:** [github.com/i0natan/nodebestpractices](https://github.com/i0natan/nodebestpractices/blob/master/sections/projectstructre/separateexpress.md)

### Q6: What is the difference between process.nextTick() and setImmediate() ?

Topic: **Node.js**  
**Difficulty: ⭐⭐⭐⭐⭐**

The difference between process.nextTick() and setImmediate() is that process.nextTick() defers the execution of an action till the next pass around the event loop or it simply calls the callback function once the ongoing execution of the event loop is finished whereas setImmediate()executes a callback on the next cycle of the event loop and it gives back to the event loop for executing any I/O operations.

🔗**Source:** [codingdefined.com](https://www.codingdefined.com/2017/04/top-20-interview-questions-on-nodejs.html)

Consider the code:

async function check(req, res) { try { const a = await someOtherFunction(); const b = await somethingElseFunction(); res.send("result") } catch (error) { res.send(error.stack); }}

Rewrite the code sample without try/catch block.

**Answer:**

async function getData(){ const a = await someFunction().catch((error)=>console.log(error)); const b = await someOtherFunction().catch((error)=>console.log(error)); if (a && b) console.log("some result")}

or if you wish to know which specific function caused error:

async function loginController() { try { const a = await loginService(). catch((error) => { throw new CustomErrorHandler({ code: 101, message: "a failed", error: error }) }); const b = await someUtil(). catch((error) => { throw new CustomErrorHandler({ code: 102, message: "b failed", error: error }) }); //someoeeoe if (a && b) console.log("no one failed") } catch (error) { if (!(error instanceof CustomErrorHandler)) { console.log("gen error", error) } }}

🔗**Source:** [medium.com](https://medium.com/tech-buddy/async-await-without-try-catch-in-javascript-fdd38abf7e90)

## Node.js Interview Questions

## 1. What is Node.js?

Wikipedia defines Node.js as “an open-source, cross-platform runtime environment for developing server-side Web applications.” It is essentially server side scripting which is used to build scalable programs.

Features of Node JS

|  |  |
| --- | --- |
| **Features** | **Description** |
| ***Fast*** | Node.js is built on Google Chrome’s V8 JavaScript Engine which makes its library very fast in code execution |
| ***Asynchronous*** | Node.js based server never waits for an API to return data thus making it asynchronous |
| ***Scalable*** | It is highly scalable because of its event mechanism which helps the server to respond in a non-blocking way |
| ***Open Source*** | Node.js has an extensive open source community which has contributed in producing some excellent modules to add additional capabilities to Node.js applications |
| ***No Buffering*** | Node.js applications simply output the data in chunks and never buffer any data. |

## 2. What is the relation of Node.js with JavaScript?

* Though Node.js is not a JavaScript framework, many of its modules are written in JavaScript.
* It allows the developers to create new modules in JavaScript.
* Node.js is a virtual machine that leverages JavaScript as its scripting language to achieve high output.

## 3. What is the fundamental difference between Node.js and Ajax?

While Ajax is a client-side technology, Node.js is a server-side JavaScript environment.

## 4. Explain the term I/O in the context of Node.js.

I/O stands for input and output. It is used to access anything outside of the application. I/O gets loaded on to the machine memory in order to run programs after the application is fired up.

## 5. State where Node.js can be used.

* Web apps
* Network applications
* Distributed systems
* General purpose applications

## 6. Which are the two types of API functions in Node.js?

Asynchronous, non-blocking functions and Synchronous, blocking functions.

## 7. List the tools and IDEs that are used for Node.js.

* Atom
* Nodeclipse Enide Studio
* JetBrains WebStorm
* JetBrains InteliJ IDEA
* MS Visual Studio
* NoFLo

## 8. Explain the role of the Callback function in Node.js.

In Node.js, the Callback function is used to cater to multiple requests made to the server. If there is a large file that is expected to take the server a long time to process, the Callback function can be invoked to ensure that other requests to the server are uninterrupted.

## 9. What is an error-first callback?

Error-first callbacks are essentially used to pass errors and data. By default, the first argument is always an error object, where the user needs to check if something is wrong.

## 10. In the context of Node.js, differentiate between operational and programmer errors.

Operational errors are not real errors. They are system errors, for example “request timeout” or “hardware error”. Programmer errors are actual bugs in the code.

## 11. Which is the framework that is most commonly used in Node.js?

The most commonly used Node.js framework is “Express”.

## 12. Define “event-driven programming.

It is essentially a programming paradigm where the program flow is characterized by events such as messages from other programs.

## 13. What are the two sections of event-driven programming?

Event Selection and Event Handling are two sections of event-driven programming.

## 14. List some of the big advantages of using Node.js.

* Ability to build scalable programs
* Increased concurrency
* Asynchronous capabilities

## 15. Explain the Control Flow function.

It is a generic piece of code that runs concurrently between several asynchronous function calls.

## 16. List the steps involved in the Control Flow function.

Control the order of execution à Collect data à Limit concurrency à Call the next program step

## 17. Can a user access DOM in a Node?

No, you cannot access DOM.

## 18. In Node.js, how do you access the last expression?

We have to use the underscore (\_) character to access the last expression.

## 19. In Node.js, which command is used to import external libraries?

A command called “require” is used for importing external libraries.

## 20. What is the biggest drawback of Node.js?

The biggest drawback is the fact that it is challenging to have one process with a single thread to scale up on multi core servers.

*Got a question for us? Please mention it in the comments section and we will get back to you.*

**What is Node.js? What is it used for?**

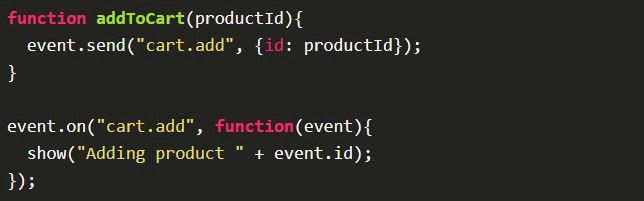
Node.js is a run-time JavaScript environment built on top of Chrome’s V8 engine. It uses an event-driven, non-blocking I/O model. It is lightweight and so efficient. Node.js has a package ecosystem called **npm**.

Node.js can be used to build different types of applications such as web application, real-time chat application, REST API server etc. However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET. Node.js was developed by Ryan Dahl in 2009.

**Q2. What is Event-driven programming?**

Event-driven programming is building our application based on and respond to events. When an event occurs, like click or keypress, we are running a callback function which is registered to the element for that event.

Event driven programming follows mainly a publish-subscribe pattern.



**Q3. What is *Event loop* in Node.js work? And How does it work?**

The *Event loop* handles all async callbacks. Node.js (or JavaScript) is a single-threaded, event-driven language. This means that we can attach listeners to events, and when a said event fires, the listener executes the callback we provided.

Whenever we are call setTimeout, http.get and fs.readFile, Node.js runs this operations and further continue to run other code without waiting for the output. When the operation is finished, it receives the output and runs our callback function.

So all the callback functions are queued in an loop, and will run one-by-one when the response has been received.

**Q4. What is REPL in Node.js?**

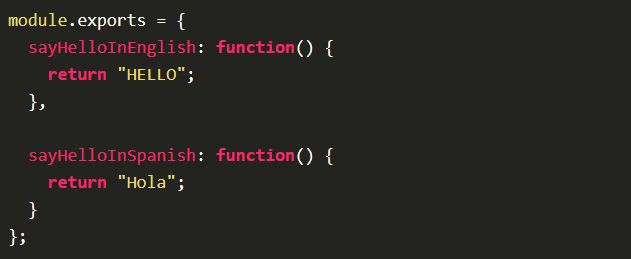
REPL means Read-Eval-Print-Loop. It is a virtual environment that comes with Node.js. We can quickly test our JavaScript code in the Node.js REPL environment.

To launch the REPL in Node.js, just open the command prompt and type node. It will change the prompt to > in Windows and MAC.

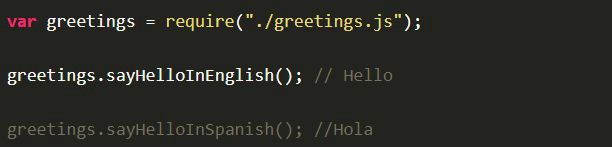
Now we can type and run our JavaScript easily. For example, if we type 10 + 20, it will print 30 in the next line.

**Q5. What is the purpose of module.exports in Node.js?**

A module encapsulates related code into a single unit of code. This can be interpreted as moving all related functions into a file. Imagine that we created a file called greetings.js and it contains the following two functions:



In the above code, module.exports exposes two functions to the outer world. We can import them in another file as follow:



**Q6. What is the difference between Asynchronous and Non-blocking?**

Asynchronous literally means not synchronous. We are making HTTP requests which are asynchronous, means we are not waiting for the server response. We continue with other block and respond to the server response when we received.

The term Non-Blocking is widely used with IO. For example non-blocking read/write calls return with whatever they can do and expect caller to execute the call again. Read will wait until it has some data and put calling thread to sleep.

**Q7. What is Tracing in Node.js?**

Tracing provides a mechanism to collect tracing information generated by V8, Node core and userspace code in a log file. Tracing can be enabled by passing the --trace-events-enabled flag when starting a Node.js application.



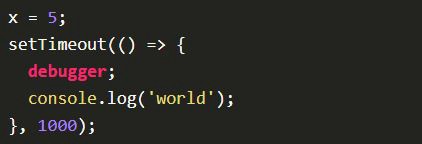
The set of categories for which traces are recorded can be specified using the --trace-event-categories flag followed by a list of comma separated category names. By default the node and v8 categories are enabled.

Running Node.js with tracing enabled will produce log files that can be opened in the chrome://tracing tab of Chrome.

**Q8. How will you debug an application in Node.js?**

Node.js includes a debugging utility called debugger. To enable it start the Node.js with the debug argument followed by the path to the script to debug.

Inserting the statement debugger; into the source code of a script will enable a breakpoint at that position in the code:



**Q9. Difference between setImmediate() vs setTimeout()**

setImmediate() and setTimeout() are similar, but behave in different ways depending on when they are called.

* setImmediate() is designed to execute a script once the current poll (event loop) phase completes.
* setTimeout() schedules a script to be run after a minimum threshold in ms has elapsed.

The order in which the timers are executed will vary depending on the context in which they are called. If both are called from within the main module, then timing will be bound by the performance of the process.

**Q10. What is process.nextTick()**

setImmediate() and setTimeout() are based on the event loop. But process.nextTick() technically not part of the event loop. Instead, the nextTickQueue will be processed after the current operation completes, regardless of the current phase of the event loop.

Thus, any time you call process.nextTick() in a given phase, all callbacks passed to process.nextTick() will be resolved before the event loop continues.

**Q11. What is package.json? What is it used for?**

This file holds various metadata information about the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

Some of the fields are: name, name, description, author and dependencies.

When someone installs our project through npm, all the dependencies listed will be installed as well. Additionally, if someone runs npm install in the root directory of our project, it will install all the dependencies to ./node\_modules directory.

**Q12. What is libuv?**

libuv is a multi-platform support library with a focus on asynchronous I/O.It was primarily developed for use by Node.js, but it’s also used by Luvit, Julia, pyuv, and others.

When the node.js project began in 2009 as a JavaScript environment decoupled from the browser, it is using Google’s V8 and Marc Lehmann’s libev, node.js combined a model of I/O – evented – with a language that was well suited to the style of programming; due to the way it had been shaped by browsers. As node.js grew in popularity, it was important to make it work on Windows, but libev ran only on Unix. libuv was an abstraction around libev or IOCP depending on the platform, providing users an API based on libev. In the node-v0.9.0 version of libuv libev was removed.

Some of the features of libuv are:

* Full-featured event loop backed by epoll, kqueue, IOCP, event ports.
* Asynchronous TCP and UDP sockets
* Asynchronous file and file system operations
* Child processes
* File system events

**Q13. What are some of the most popular modules of Node.js?**

There are many most popular, most starred or most downloaded modules in Node.js. Some of them are:

* express
* async
* browserify
* socket.io
* bower
* gulp
* grunt

**Q14. What is EventEmitter in Node.js?**

All objects that emit events are instances of the EventEmitter class. These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.

When the EventEmitter object emits an event, all of the functions attached to that specific event are called *synchronously*.



**Q15. What is Streams in Node.js?**

Streams are pipes that let you easily read data from a source and pipe it to a destination. Simply put, a stream is nothing but an EventEmitter and implements some specials methods. Depending on the methods implemented, a stream becomes Readable, Writable, or Duplex (both readable and writable).

For example, if we want to read data from a file, the best way to do it from a stream is to listen to data event and attach a callback. When a chunk of data is available, the readable stream emits a data event and your callback executes. Take a look at the following snippet:



Types of streams are: Readable, Writable, Duplex and Transform.

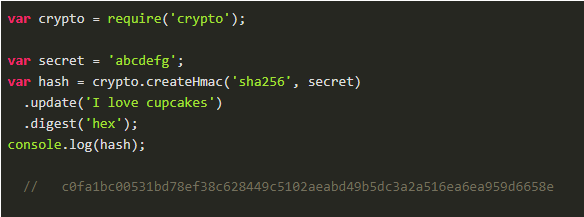
**Q16**. **What is the difference between readFile vs createReadStream in Node.js?**

**readFile** — is for asynchronously reads the entire contents of a file. It will read the file completely into memory before making it available to the User. readFileSync is synchronous version of readFile.

**createReadStream** — It will read the file in chunks of the default size 64 kb which is specified before hand.

**Q17. What is crypto in Node.js? How do you cipher the secured information in Node.js?**

The crypto module in Node.js provides cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign and verify functions.



**Q18. What is the use of Timers is Node.js?**

The Timers module in Node.js contains functions that execute code after a set period of time. Timers do not need to be imported via require(), since all the methods are available globally to emulate the browser JavaScript API.

The Node.js API provides several ways of scheduling code to execute at some point after the present moment. The functions below may seem familiar, since they are available in most browsers, but Node.js actually provides its own implementation of these methods.

Node.js Timer provides setTimeout(), setImmediate() and setInterval.

**Q19. What is the use of DNS module in Node.js?**

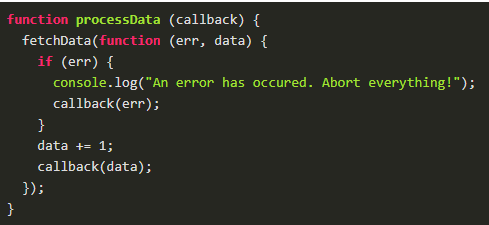
dns module which provide underlying system's name resolution and DNS look up facilities. DNS module consists of an asynchronous network wrapper.

The most commonly used functions in DNS module are:

* **dns.lookup(adress, options, callback)** - The dns lookup method takes any website address as its first parameter and returns the corresponding first IPV4 or IPV6 record. The options parameter can be an integer or object. If no options are provided both IPV4 and IPV6 are valid inputs. The third parameter is the callback functions.
* **dns.lookupservice(address, port, callback)** - This function converts any physical address such as “www.knowledgehills.com” to array of record types. The record types are specified by the second parameter “rrbyte”. Finally the third method is the callback function.
* **dns.getServers()** - This function returns an array of IP address strings, formatted according to rfc5952, that are currently configured for DNS resolution. A string will include a port section if a custom port is used.
* **dns.setServers()** - This function sets the IP address and port of servers to be used when performing DNS resolution. The dns.setServers() method must not be called while a DNS query is in progress.

**Q20. What is a Callback function in Node.js?**

Node.js, being an asynchronous platform, doesn’t wait around for things like file I/O to finish — Node.js uses callbacks. A callback is a function called at the completion of a given task; this prevents any blocking, and allows other code to be run in the meantime.



Callbacks are the foundation of Node.js. Callbacks give us an interface with which to say, “and when you’re done doing that, do all this.” This allows us to have as many IO operations as our OS can handle happening at the same time. For example, in a web server with hundreds or thousands of pending requests with multiple blocking queries, performing the blocking queries asynchronously gives you the ability to be able to continue working and not just sit still and wait until the blocking operations come back.

**Q21. What are the security mechanisms available in Node.js?**

We can secure our Node.js application in the following ways:

**Authentication** — Authentication is one of the primary security stages at which user is identified as permitted to access the application at all. Authentication verifies the user’s identity through one or several checks. In Node.js, authentication can be either session-based or token-based. In session-based authentication, the user’s credentials are compared to the user account stored on the server and, in the event of successful validation, a session is started for the user. Whenever the session expires, the user needs to log in again. In token-based authentication, the user’s credentials are applied to generate a string called a token which is then associated with the user’s requests to the server.

**Error Handling** — Usually, the error message contains the explanation of what’s actually gone wrong for the user to understand the reason. At the same time, when the error is related to the application code syntax, it can be set to display the entire log content on the frontend. For an experienced hacker, the log content can reveal a lot of sensitive internal information about the application code structure and tools used within the software.

**Request Validation** — Another aspect which has to be considered, while building a secure Node.js application, is a validation of requests or, in other words, a check of the incoming data for possible inconsistencies. It may seem that invalid requests do not directly affect the security of a Node.js application, however, they may influence its performance and robustness. Validating the incoming data types and formats and rejecting requests not conforming to the set rules can be an additional measure of securing your Node.js application.

**Node.js Security Tools and Best Practices** — We can use tools like **helmet**(protects our application by setting HTTP headers), **csurf** (validates tokens in incoming requests and rejects the invalid ones), **node rate limiter** (controls the rate of repeated requests. This function can protect you from brute force attacks) and **cors** (enables cross-origin resource sharing).

**Q22. What is the passport in Node.js?**

Passport.js is a simple, unobtrusive Node.js authentication middleware for Node.js. Passport.js can be dropped into any Express.js-based web application.

Passport recognizes that each application has unique authentication requirements. Authentication mechanisms, known as strategies, are packaged as individual modules. Applications can choose which strategies to employ, without creating unnecessary dependencies.

By default, if authentication fails, Passport will respond with a 401 Unauthorized status, and any additional route handlers will not be invoked. If authentication succeeds, the next handler will be invoked and the req.user property will be set to the authenticated user.

For more Node JS Interview Questions and answer install our Android App:

#### What is an error-first callback?

Error-first callbacks are used to pass errors and data as well. You have to pass the error as the first parameter, and it has to be checked to see if something went wrong. Additional arguments are used to pass data.

fs.readFile(filePath, **function**(err, data) { **if** (err) { *// handle the error, the return is important here* *// so execution stops here* **return** console.log(err) } *// use the data object* console.log(data)})

#### How can you avoid callback hells?

There are lots of ways to solve the issue of callback hells:

* **modularization**: break callbacks into independent functions
* use a **control flow library**, like [async](https://www.npmjs.com/package/async)
* use **generators with Promises**
* use **async/await** *(note that it is only available in the latest v7 release and not in the LTS version -* [*you can read our experimental async/await how-to here*](https://blog.risingstack.com/async-await-node-js-7-nightly/)*)*

[**Q: How to avoid callback hells? A: modularization, control flow libraries, generators with promises, async/await**](https://twitter.com/share?text=Q%3AHow%20to%20avoid%20callback%20hells%3F%20A%3A%20modularization%2C%20control%20flow%20libraries%2C%20generators%20with%20promises%2C%20async%2Fawait;url=https://blog.risingstack.com/node-js-interview-questions-and-answers-2017)

[CLICK TO TWEET](https://twitter.com/share?text=Q%3A%20How%20to%20avoid%20callback%20hells%3F%20A%3A%20modularization%2C%20control%20flow%20libraries%2C%20generators%20with%20promises%2C%20async%2Fawait;url=https://blog.risingstack.com/node-js-interview-questions-and-answers-2017)

#### What are Promises?

Promises are a concurrency primitive, first described in the 80s. Now they are part of most modern programming languages to make your life easier. Promises can help you better handle async operations.

An example can be the following snippet, which after 100ms prints out the resultstring to the standard output. Also, note the catch, which can be used for error handling. Promises are chainable.

**new** Promise((resolve, reject) => { setTimeout(() => { resolve('result') }, 100)}) .then(console.log) .catch(console.error)

#### What tools can be used to assure consistent style? Why is it important?

When working in a team, consistent style is important, so team members can modify more projects easily, without having to get used to a new style each time.

Also, it can help eliminate programming issues using static analysis.

Tools that can help:

* [ESLint](http://eslint.org/)
* [Standard](http://standardjs.com/)

If you’d like to be even more confident, I suggest you to learn and embrace the [JavaScript Clean Coding](https://blog.risingstack.com/javascript-clean-coding-best-practices-node-js-at-scale/) principles as well!

#### What's a stub? Name a use case!

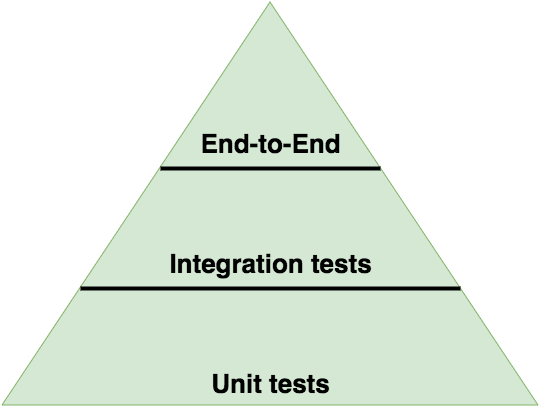
Stubs are functions/programs that simulate the behaviors of components/modules. Stubs provide canned answers to function calls made during test cases.

An example can be writing a file, without actually doing so.

**var** fs = require('fs')**var** writeFileStub = sinon.stub(fs, 'writeFile', **function** (path, data, cb) { **return** cb(null)})expect(writeFileStub).to.be.calledwriteFileStub.restore()

#### What's a test pyramid? Give an example!

A test pyramid describes the ratio of how many unit tests, integration tests and end-to-end test you should write.



An example for an HTTP API may look like this:

* lots of low-level unit tests for models *(dependencies* ***are stubbed****)*,
* fewer integration tests, where you check how your models interact with each other *(dependencies* ***are not stubbed****)*,
* less end-to-end tests, where you call your actual endpoints *(dependencies* ***are not stubbed****)*.

#### What's your favorite HTTP framework and why?

There is no right answer for this. The goal here is to understand how deeply one knows the framework she/he uses. Tell what are the pros and cons of picking that framework.

#### When are background/worker processes useful? How can you handle worker tasks?

Worker processes are extremely useful if you'd like to do data processing in the background, like sending out emails or processing images.

There are lots of options for this like [RabbitMQ](https://www.rabbitmq.com/) or [Kafka](https://kafka.apache.org/).

#### How can you secure your HTTP cookies against XSS attacks?

XSS occurs when the attacker injects executable JavaScript code into the HTML response.

To mitigate these attacks, you have to set flags on the set-cookie HTTP header:

* **HttpOnly** - this attribute is used to help prevent attacks such as cross-site scripting since it does not allow the cookie to be accessed via JavaScript.
* **secure** - this attribute tells the browser to only send the cookie if the request is being sent over HTTPS.

So it would look something like this: Set-Cookie: sid=<cookie-value>; HttpOnly. If you are using Express, with [express-cookie session](https://github.com/expressjs/cookie-session#cookie-options), it is working by default.

#### How can you make sure your dependencies are safe?

When writing Node.js applications, **ending up with hundreds or even thousands of dependencies can easily happen**.  
For example, if you depend on Express, you depend on [27 other modules](https://github.com/expressjs/express/blob/master/package.json#L29) directly, and of course on those dependencies' as well, so manually checking all of them is not an option!

The only option is to automate the update / security audit of your dependencies. For that there are free and paid options:

* npm outdated
* [Trace by RisingStack](https://trace.risingstack.com/)
* [NSP](https://nodesecurity.io/)
* [GreenKeeper](https://greenkeeper.io/)
* [Snyk](https://snyk.io/)

## Node.js Interview Puzzles

The following part of the article is useful if you’d like to prepare for an interview that involves puzzles, or tricky questions.

### What's wrong with the code snippet?

**new** Promise((resolve, reject) => { **throw** **new** Error('error')}).then(console.log)

#### The Solution

As there is no catch after the then. This way the error will be a silent one, there will be no indication of an error thrown.

To fix it, you can do the following:

**new** Promise((resolve, reject) => { **throw** **new** Error('error')}).then(console.log).catch(console.error)

If you have to debug a huge codebase, and you don't know which Promise can potentially hide an issue, you can use the unhandledRejection hook. It will print out all unhandled Promise rejections.

process.on('unhandledRejection', (err) => { console.log(err)})

### What's wrong with the following code snippet?

**function** **checkApiKey** (apiKeyFromDb, apiKeyReceived) { **if** (apiKeyFromDb === apiKeyReceived) { **return** true } **return** false}

#### The Solution

When you compare security credentials it is crucial that you don't leak any information, so you have to make sure that you compare them in fixed time. If you fail to do so, your application will be vulnerable to [timing attacks](https://en.wikipedia.org/wiki/Timing_attack).

But why does it work like that?

**V8, the JavaScript engine used by Node.js, tries to optimize the code you run from a performance point of view.** It starts comparing the strings character by character, and once a mismatch is found, it stops the comparison operation. **So the longer the attacker has right from the password, the more time it takes.**

To solve this issue, you can use the npm module called [cryptiles](https://www.npmjs.com/package/cryptiles).

**function** **checkApiKey** (apiKeyFromDb, apiKeyReceived) { **return** cryptiles.fixedTimeComparison(apiKeyFromDb, apiKeyReceived)}

### What's the output of following code snippet?

Promise.resolve(1) .then((x) => x + 1) .then((x) => { **throw** **new** Error('My Error') }) .catch(() => 1) .then((x) => x + 1) .then((x) => console.log(x)) .catch(console.error)

#### The Answer

The short answer is 2 - however with this question **I'd recommend asking the candidates to explain what will happen line-by-line to understand how they think**. It should be something like this:

1. A new Promise is created, that will resolve to 1.
2. The resolved value is incremented with 1 (so it is 2 now), and returned instantly.
3. The resolved value is discarded, and an error is thrown.
4. The error is discarded, and a new value (1) is returned.
5. The execution did not stop after the catch, but before the exception was handled, it continued, and a new, incremented value (2) is returned.
6. The value is printed to the standard output.
7. This line won't run, as there was no exception.

What is Node.js? Where can you use it?

Node.js is a server side scripting based on Google’s V8 JavaScript engine. It is used to build scalable programs especially web applications that are computationally simple but are frequently accessed.

You can use [Node.js in developing](https://www.simplilearn.com/developing-java-and-node-js-application-with-mongodb-tutorial-video) I/O intensive web applications like video streaming sites. You can also use it for developing: Real-time web applications, Network applications, General-purpose applications and Distributed systems.

Why use Node.js?

Node.js makes building scalable network programs easy. Some of its advantages include:

* It is generally fast
* It almost never blocks
* It offers a unified programming language and data type
* Everything is asynchronous
* It yields great concurrency

What are the features of Node.js?

Node.js is a single-threaded but highly scalable system that utilizes JavaScript as its scripting language. It uses asynchronous, event-driven I/O instead of separate processes or threads. It is able to achieve high output via single-threaded event loop and non-blocking I/O.

How else can the JavaScript code below be written using Node.Js to produce the same output?

console.log("first");  
setTimeout(function() {  
 console.log("second");  
}, 0);  
console.log("third");

Output:

first  
third  
second

In Node.js version 0.10 or higher, setImmediate(fn) will be used in place of setTimeout(fn,0) since it is faster. As such, the code can be written as follows:

console.log("first");  
setImmediate(function(){  
 console.log("second");  
});  
console.log("third");

How do you update NPM to a new version in Node.js?

You use the following commands to update NPM to a new version:

$ sudo npm install npm -g  
/usr/bin/npm -> /usr/lib/node\_modules/npm/bin/npm-cli.js  
[npm@2.7.1](mailto:npm@2.7.1) /usr/lib/node\_modules/npm

Why is Node.js Single-threaded?

Node.js is single-threaded for async processing. By doing async processing on a single-thread under typical web loads, more performance and scalability can be achieved as opposed to the typical thread-based implementation.

Explain callback in Node.js.

A callback function is called at the completion of a given task. This allows other code to be run in the meantime and prevents any blocking. Being an asynchronous platform, Node.js heavily relies on callback. All APIs of Node are written to support callbacks.

What is callback hell in Node.js?

Callback hell is the result of heavily nested callbacks that make the code not only unreadable but also difficult to maintain. For example:

query("SELECT clientId FROM clients WHERE clientName='picanteverde';", function(id){  
 query("SELECT \* FROM transactions WHERE clientId=" + id, function(transactions){  
 transactions.each(function(transac){  
 query("UPDATE transactions SET value = " + (transac.value\*0.1) + " WHERE id=" + transac.id, function(error){  
 if(!error){  
 console.log("success!!");  
 }else{  
 console.log("error");  
 }  
 });  
 });  
 });  
});

How do you prevent/fix callback hell?

The three ways to prevent/fix callback hell are:

* Handle every single error
* Keep your code shallow
* Modularize – split the callbacks into smaller, independent functions that can be called with some parameters then joining them to achieve desired results.

The first level of improving the code above might be:

var logError = function(error){  
 if(!error){  
 console.log("success!!");  
 }else{  
 console.log("error");  
 }  
 },  
 updateTransaction = function(t){  
 query("UPDATE transactions SET value = " + (t.value\*0.1) + " WHERE id=" + t.id, logError);  
 },  
 handleTransactions = function(transactions){  
 transactions.each(updateTransaction);  
 },  
 handleClient = function(id){  
 query("SELECT \* FROM transactions WHERE clientId=" + id, handleTransactions);  
 };

query("SELECT clientId FROM clients WHERE clientName='picanteverde';",handleClient);

You can also use Promises, Generators and Async functions to fix callback hell.

Explain the role of REPL in Node.js.

As the name suggests, REPL (Read Eval print Loop) performs the tasks of – Read, Evaluate, Print and Loop. The REPL in Node.js is used to execute ad-hoc Javascript statements. The REPL shell allows entry to javascript directly into a shell prompt and evaluates the results. For the purpose of testing, debugging, or experimenting, REPL is very critical.

11. Name the types of API functions in Node.js.

There are two types of functions in Node.js.:

* Blocking functions - In a blocking operation, all other code is blocked from executing until an I/O event that is being waited on occurs. Blocking functions execute synchronously

For example:  
const fs = require('fs');  
const data = fs.readFileSync('/file.md'); // blocks here until file is read  
console.log(data);  
// moreWork(); will run after console.log

The second line of code blocks the execution of additional JavaScript until the entire file is read. moreWork () will only be called after Console.log

* Non-blocking functions - In a non-blocking operation, multiple I/O calls can be performed without the execution of the program being halted. Non-blocking functions execute asynchronously.

For example:

const fs = require('fs');  
fs.readFile('/file.md', (err, data) => {  
 if (err) throw err;  
 console.log(data);  
});  
// moreWork(); will run before console.log

Since fs.readFile () is non-blocking, moreWork () does not have to wait for the file read to complete before being called. This allows for higher throughput.

Which is the first argument typically passed to a Node.js callback handler?

Typically, the first argument to any callback handler is an optional error object. The argument is null or undefined if there is no error.

Error handling by a typical callback handler could be as follows:

function callback(err, results) {  
 // usually we'll check for the error before handling results  
 if(err) {  
 // handle error somehow and return  
 }  
 // no error, perform standard callback handling  
}

What are the functionalities of NPM in Node.js?

NPM (Node package Manager) provides two functionalities:

* Online repository for Node.js packages
* Command line utility for installing packages, version management and dependency management of Node.js packages

14. What is the difference between Node.js and Ajax?

Node.js and Ajax (Asynchronous JavaScript and XML) are the advanced implementation of JavaScript. They all serve completely different purposes.

Ajax is primarily designed for dynamically updating a particular section of a page’s content, without having to update the entire page.

Explain chaining in Node.js.

Chaining is a mechanism whereby the output of one stream is connected to another stream creating a chain of multiple stream operations.

16. What are “streams” in Node.js? Explain the different types of streams present in Node.js.

Streams are objects that allow reading of data from the source and writing of data to the destination as a continuous process.

There are four types of streams.

* to facilitate the reading operation
* to facilitate the writing operation
* to facilitate both read and write operations
* is a form of Duplex stream that performs computations based on the available input

17. What are exit codes in Node.js? List some exit codes.

Exit codes are specific codes that are used to end a “process” (a global object used to represent a node process).

Examples of exit codes include:

* Unused
* Uncaught Fatal Exception
* Fatal Error
* Non-function Internal Exception Handler
* Internal Exception handler Run-Time Failure
* Internal JavaScript Evaluation Failure

18. What are Globals in Node.js?

Three keywords in Node.js constitute as Globals. These are:

* Global – it represents the Global namespace object and acts as a container for all otherobjects.
* Process – It is one of the global objects but can turn a synchronous function into an async callback. It can be accessed from anywhere in the code and it primarily gives back information about the application or the environment.
* Buffer – it is a class in Node.js to handle binary data.

19. What is the difference between AngularJS and Node.js?

Angular.JS is a web application development framework while Node.js is a runtime system.

20. Why is consistent style important and what tools can be used to assure it?

Consistent style helps team members modify projects easily without having to get used to a new style every time. Tools that can help include Standard and ESLint.